CS 7810 - Knowledge Representation and Reasoning (for the Semantic Web)

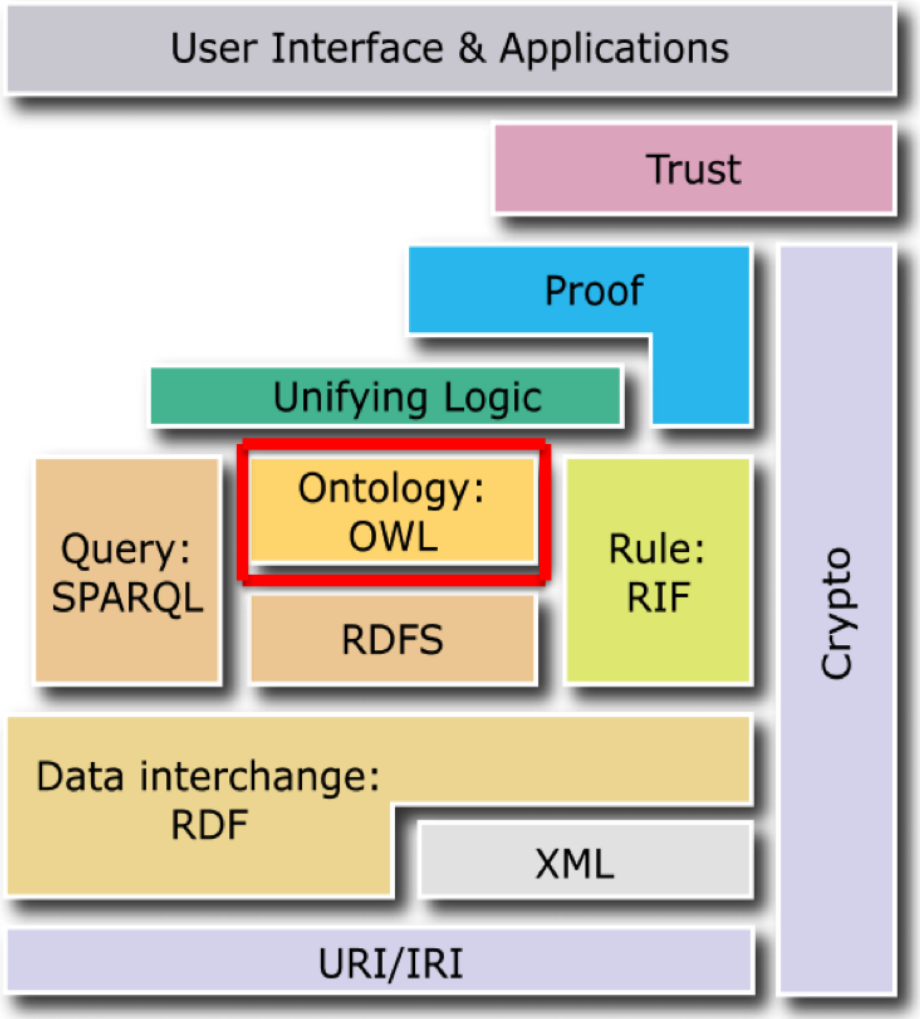
06 - The Web Ontology Language (OWL)

Adila Krisnadhi

Data Semantics Lab
Wright State University, Dayton, OH

October 20, 2016

Outline


User Interface & Applications
Trust
Proof
Unifying Logic
Query: SPARQL
Ontology: OWL
Rule: RIF
RDFS
Crypto
Data interchange: RDF
XML
URI/IRI


W3C
OWL

Acknowledgements

Materials in this presentation are adapted from:

- Markus Krötzsch, "OWL Syntax and Intuition", slides for Foundations of Semantic Web Technologies course, TU Dresden, May 14, 2014.
- Markus Krötzsch, "OWL & Description Logics", slides for Foundations of Semantic Web Technologies course, TU Dresden, May 16, 2014.
- Markus Krötzsch, "OWL Syntax and Semantics", slides for Foundations of Semantic Web Technologies course, TU Dresden, May 16, 2014.
- Axel Polleres, "Unit 5: OWL, OWL 2, SPARQL+OWL", slides for Semantic Web Technologies course, TU Vienna, June 06, 2012.

Outline

What is Ontology?

- **Philosophy**: ontology = "The study of being"
  - singular term (no 'ontologies')
  - first mentioned in 17th century
  - found in writings of Socrates, Aristotle, Aquinas, Descartes, Kant, Hegel, Wittgenstein, Heidegger, Quine, ...
- **Computer science**: (Gruber 1993)
  "an ontology is a

formal specification	$\rightsquigarrow$ machine interpretable
of a **shared**	$\rightsquigarrow$ follows from a consensus
conceptualization	$\rightsquigarrow$ describes relevant notions
of a **domain** of interest"	$\rightsquigarrow$ corresponds to a topic/theme

Ontology Language Features[1]

- concepts/classes: "anything about which something is said"
  - could be description of a task, function, entity, etc.
  - constructors of complex concepts/classes: logical operators, quantifiers, etc.
  - datatypes: classes/concepts for concrete literal values
- conceptual hierarchies (taxonomies, inheritance?)
- relations
- axioms or logical assertions involving concepts and relations
- instantiations of classes by individuals
- precise semantics and inference mechanisms

[1] Corcho, Ó., Gómez-Pérez, A., "A Roadmap to Ontology Specification Languages", EKAW 2000

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:

RDF/RDFS: Simple Ontology Languages?

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.

RDF/RDFS: Simple Ontology Languages?

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).

RDF/RDFS: Simple Ontology Languages?

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)
  - Characteristics of relations (domain and range of relations)

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)
  - Characteristics of relations (domain and range of relations)
  - RDFS semantics and entailments.

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)
  - Characteristics of relations (domain and range of relations)
  - RDFS semantics and entailments.

RDFS is an ontology language!

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)
  - Characteristics of relations (domain and range of relations)
  - RDFS semantics and entailments.

RDFS is an ontology language!

- Advantage: automated inference is relatively efficient/cheap

RDF/RDFS: Simple Ontology Languages?

DaSe Lab

- RDF is not quite an ontology language: every triple is a logical assertion, but can only specify relations between particular individuals.
- RDFS:
  - Every triple is a logical assertion.
  - Classes, datatypes, taxonomies (via `rdfs:subClassOf`), instantiations of classes by individuals (via `rdf:type`).
  - (Atomic) relations between particular individuals (since it subsumes RDF) via properties.
  - Hierarchy of (atomic) relations (via `rdfs:subPropertyOf`)
  - Characteristics of relations (domain and range of relations)
  - RDFS semantics and entailments.

RDFS is an ontology language!

- Advantage: automated inference is relatively efficient/cheap
- But: only appropriate for simple ontologies and not for more complex modeling $\rightsquigarrow$ Needs more a expressive language, e.g., OWL, RIF.

Ontology in RDFS: Example

```
1   ex:Pascal      rdf:type            ex:Faculty .
2   ex:Sarker      rdf:type            ex:PhDStudent .
3   ex:Sarker      rdf:type            ex:GRA .
4   ex:GRA         rdfs:subClassOf     ex:Student .
5   ex:GRA         rdfs:subClassOf     ex:Employee .
6   ex:Faculty     rdfs:subClassOf     ex:Employee .
7   ex:PhDStudent  rdfs:subClassOf     ex:Student .
8   ex:email       rdfs:range          xsd:string .
9   ex:advises     rdfs:domain         ex:Faculty .
10  ex:advises     rdfs:range          ex:Student .
11  ex:supervises  rdfs:domain         ex:Employee .
12  ex:supervises  rdfs:range          ex:Employee .
13  ex:advises     rdfs:subPropertyOf  ex:responsibleFor .
14  ex:supervises  rdfs:subPropertyOf  ex:responsibleFor .
```

- Classes: `ex:Faculty`, `ex:PhDStudent`, `ex:Student`, `ex:GRA`, `ex:Employee`
- Relations/properties: `ex:email`, `ex:advises`, `ex:supervises`, `ex:responsibleFor`
- 1,2,3 are instantiations of classes by individuals
- 4,5,6,7 form a taxonomy
- 8,9,10,11,12 are characteristics of relations
- 13,14 form a hierarchy of relations.

What can/cannot be expressed in RDFS?

Determine whether the following statements can be expressed in RDFS.

- "Every car is a vehicle."
- "The Nile is a river."
- "Mississippi river flows through the State of Louisiana."
- "Every professor advises at least two different students."
- "For every x, if x studies at y, then x is a student."
- "Every professor advises only students."
- "Pascal advises at least Sarker, Nazifa, and possibly a few others."
- "No university can be both public and private."
- "Every university is either public or private."

Outline

OWL 1

- Predecessors: DAML, OIL.
- W3C Recommendation since 2004
- Based on description logics – fragment/sublanguage of first-order logic.
- Three variants:
  - OWL Lite: fragment of OWL DL; as expressive as the description logic $\mathcal{SHIF}(D)$; reasoning is worst case ExPTime-complete.
  - OWL DL: classes and properties strictly separated; contains OWL Lite; disallow several RDFS language elements; corresponds to the description logic $\mathcal{SHOIN}(D)$; widely supported by tools; reasoning is worst case NExPTime complete.
  - OWL Full: covers OWL DL and RDFS; undecidable; limited tool support; semantics contains problematic aspects (e.g., no separation between classes, properties, and individuals)
- Reification is not supported in OWL DL $\rightsquigarrow$ RDFS is a fragment of OWL Full, but not OWL DL.

OWL 2

- Supersedes OWL 2 by adding several new logical constructs, user-defined datatypes, metamodeling via punning.
- Corresponds to the description logic $\mathcal{SROIQ}(D)$
- W3C Recommendation since 2009:
  `https://www.w3.org/TR/owl2-overview/`
- Specified also three OWL 2 "Profiles" $\rightsquigarrow$ small fragments of OWL 2 that allow much more efficient reasoning, cover important application areas, and are easily understandable to non-experts.
  - OWL 2 QL
  - OWL 2 RL
  - OWL 2 EL
- We shall only focus on OWL 2 (henceforth simply called OWL) and only mention OWL 1 only when it is necessary.

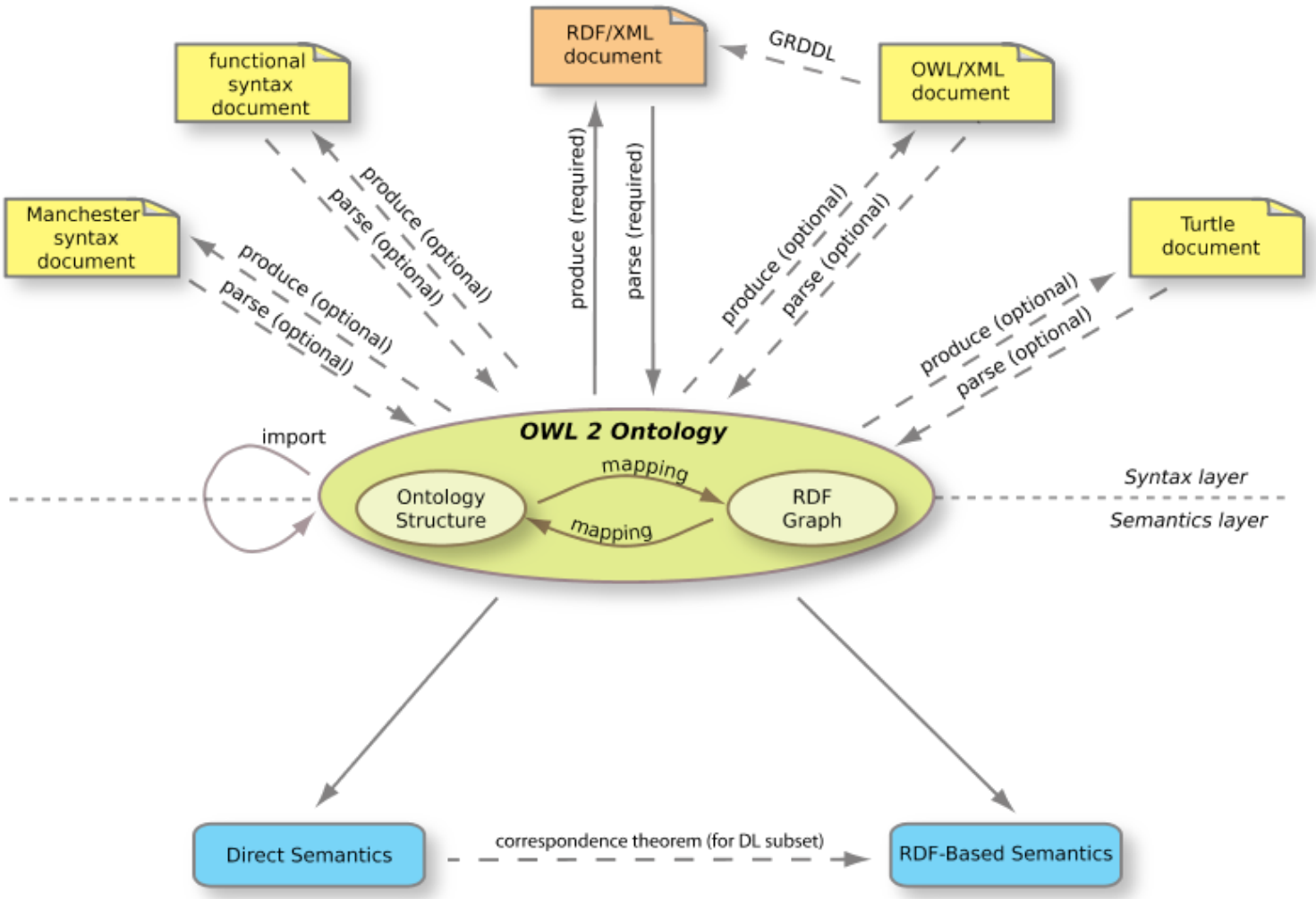


Figure: `https://www.w3.org/TR/owl2-overview/OWL2-structure2-800.png`

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.
- OWL is **not** a schema language for syntax conformance (unlike XML)

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.
- OWL is **not** a schema language for syntax conformance (unlike XML)
  - Does not provide means to specify how a document should be syntactically structured.

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.
- OWL is **not** a schema language for syntax conformance (unlike XML)
  - Does not provide means to specify how a document should be syntactically structured.
  - Cannot enforce that a particular piece of information (say, an SSN) must syntactically be present in the document.

What is (not) OWL

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.
- OWL is **not** a schema language for syntax conformance (unlike XML)
  - Does not provide means to specify how a document should be syntactically structured.
  - Cannot enforce that a particular piece of information (say, an SSN) must syntactically be present in the document.
- OWL s **not** a database framework

What is (not) OWL

DaSe Lab

- OWL is **not** a programming language: it is declarative – allows logical description of a state of affairs.
  - Tools (called **reasoners**) can be used to infer more information about such a state of affairs
  - OWL formal semantics specifies the correct answer of such inferences, but not how they are algorithmically realized.
- OWL is **not** an object-oriented modeling language, at least in the traditional sense, although some resemblance exists.
- OWL is **not** a schema language for syntax conformance (unlike XML)
  - Does not provide means to specify how a document should be syntactically structured.
  - Cannot enforce that a particular piece of information (say, an SSN) must syntactically be present in the document.
- OWL s **not** a database framework
  - Databases follow the **closed-world assumption**, while OWL 2 follows the **open-world assumption**

- **Closed-world assumption**: if a statement F is not present or cannot be inferred from the database/knowledge base, then F must be false (equivalently, the negation of F is true).
- **Open-world assumption**: if a statement F is missing or cannot be inferred from the database/knowledge base, then we cannot say anything about it (besides the fact that it cannot be inferred from the database/knowledge base). That is, we simply don't know if F is true or not.

OWL Documents

- From a logical perspective, every OWL ontology is a set of OWL axioms.
- OWL ontologies are serialized as **OWL documents** using some OWL syntax.
- OWL 2 standard provides five syntaxes: Manchester, functional, RDF/XML, OWL/XML, Turtle.
- In the literature, we also often find OWL ontologies described in description logic (DL) syntax.
- An OWL document consists of:
  - ontology header: name of the ontology, versioning information, import directives, etc.
  - the actual content of the ontology (as a set of axioms) – each axiom is constructed from the components described next.

Outline

Basic Components of OWL

- **Individuals**: correspond to actual objects from the universe.
  - **Named** individuals: IRIs representing individuals.
  - **Anonymous** individuals: denoted by a local node ID.
- **Classes**: IRIs representing a set of individuals
- **Literals**: data values such as particular strings or integers (analogous to typed RDF literals).
- **Datatypes**: IRIs representing a set of data values
- **Properties**: correspond to binary relations
  - **Object** properties: IRIs for binary relations that connect pairs of individuals
  - **Data** properties: IRIs for binary relations that connect individuals to literals
  - **Annotation** properties: IRIs representing binary relations connecting anything (ontology, axioms, IRIs) to annotations. Annotations are not part of the logical meaning of an ontology, but rather, is used for giving extra information for documenting the ontology.x
- All components given as IRIs (i.e., named individuals, classes, datatypes, properties) are called **(OWL) entity**
- An ontology may contain declarations for entities: no logical meaning and, in some cases, are optional.

Classes I

- IRIs understood as a set of individuals.
- In description logic, also called **concepts**
- Built-in classes:
  - `owl:Thing` – the set of all individuals (in DL, called the **top concept**, written using the symbol $\top$)
  - `owl:Nothing` – the empty set (in DL, called the **bottom concept**, written using the symbol $\bot$)

  Built-in classes are always implicitly declared.
  - Every class is a subclass of `owl:Thing`.
  - `owl:Nothing` is a subclass of every class.

Functional Syntax

```
SubClassOf( ex:PhDStudent ex:Student )
```

"every PhD student is a student"
`ex:PhDStudent`: the set of all PhD students (in the application domain).
`ex:Student`: the class of all students.

Class declarations:

```
Declaration( Class( ex:PhDStudent ) )
Declaration( Class( ex:Student ) )
```

Manchester syntax

```
Class: ex:Student
Class: ex:PhDStudent
  SubClassOf: ex:Student
```

DL syntax

$$\texttt{ex:PhDStudent} \sqsubseteq \texttt{ex:Student}$$

Declarations are never used in the DL syntax.

RDF/XML syntax – Turtle syntax can be obtained from this

```
<owl:Class rdf:about="http://example.org/onto#Student"/>
<owl:Class rdf:about="http://example.org/onto#PhDStudent">
 <rdfs:subClassOf rdf:resource="http://example.org/onto#Student"/>
</owl:Class>
```

`owl:Class` is **different** from `rdfs:Class`!

OWL/XML syntax

```
<Declaration> <Class IRI="#Student"/> </Declaration>
<Declaration> <Class IRI="#PhDStudent"/> </Declaration>
<SubClassOf>
    <Class IRI="#PhDStudent"/>
    <Class IRI="#Student"/>
</SubClassOf>
```

Individuals I

- Represent actual objects in the universe.
- Named individuals: those individuals identified using an IRI.
- Anonymous individuals: those only identified by a local node ID (like a blank node in RDF).

Functional syntax

```
ClassAssertion( ex:Student ex:sarker )
```

"Sarker is a student."

Declaration:

```
Declaration( NamedIndividual( ex:sarker ) )
```

Individuals II

DL syntax

ex:Student(ex:sarker)

Manchester syntax

```
Individual: ex:sarker
    Types:  ex:Student
```

RDF/XML syntax

```
<owl:NamedIndividual rdf:about="http://example.org/onto#sarker">
    <rdf:type rdf:resource="http://example.org/onto#Student"/>
</owl:NamedIndividual>
```

Object Properties I

- IRIs connecting two individuals (named or anonymous)

Functional syntax

```
Declaration( ObjectProperty( ex:studiesAt ) )
ObjectPropertyAssertion( ex:studiesAt ex:sarker ex:wright_state )
```

"Sarker studies at Wright State"

DL syntax

$$\texttt{ex:studiesAt}(\texttt{ex:sarker}, \texttt{ex:wright\_state})$$

Object Properties II

Manchester syntax

```
Individual: ex:sarker
  Facts:  ex:studiesAt  ex:wright_state
```

RDF/XML

```
<owl:ObjectProperty rdf:about="http://example.org/onto#studiesAt"/>
<owl:NamedIndividual rdf:about="http://example.org/onto#sarker">
   <studiesAt rdf:resource="http://example.org/onto#wsu"/>
</owl:NamedIndividual>
```

Data Properties I

- IRIs connecting individuals to literals.

Functional syntax

```
Declaration( DataProperty( ex:hasName ) )
DataPropertyAssertion(ex:hasName ex:sarker "Sarker"^^xsd:string)
```

"Sarker has a name 'Sarker' "

RDF/XML

```
<owl:DatatypeProperty rdf:about="http://example.org/onto#hasName"/>
<owl:NamedIndividual rdf:about="http://example.org/onto#sarker">
  <rdf:type rdf:resource="http://example.org/onto#Student"/>
  <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
     Sarker
  </hasName>
</owl:NamedIndividual>
```

Outline

Axioms

Every OWL ontology is a set of OWL axioms.

An OWL axiom can be of the following form (we've seen some examples already):

- **Declaration**
- **Class axiom**: subclassing, equivalence, disjointness, disjoint union
- **Object property axiom**: subproperties, equivalence, disjointness, inverse object properties, domain, range, functionality, inverse functionality, reflexivity, irreflexivity, symmetry, assymmetry, transitivity
- **Data property axiom**: subproperties, equivalence, disjointness, domain, range, functionality
- **(ABox) assertion**: individual equality & inequality, class assertion, (positive/negative) (object/data) property assertion
- **HasKey axiom**
- **Annotation axiom**: subproperties, domain, range – does not affect logical meaning.

Work in Your Project Group

- Create three example sentences that describe your domain of choice. For example, your sentence may look like as follows:
  - "A Pizza can either be a vegetarian pizza or a non-vegetarian one."
  - "Every thin and crispy pizza base is a pizza base."
  - "Pizza bases are different from pizza toppings."
  - "If x has a topping y, then x has ingredient y."
  - "If x has ingredient y, then y is an ingredient of x."
  - "The relationship of having ingredient is transitive."
  - "Every pizza has at most one pizza base."
  - "If a pizza has a topping, then that topping must be a pizza topping."
  - "Something that has a pizza topping as its topping must be a pizza."
  - "Margherita pizzas are pizzas that have a mozzarella topping and a tomato topping as their toppings."
  - "Margherita pizzas have tomato and mozzarella toppings as their only toppings."
  - "Special pizzas have exactly three toppings."
  - "High calorie pizzas have calorific content value of at least 400."
  - "Each type of pizza has exactly one calorific content value."
- Exchange your set of sentences with another group in the class.
- By consulting examples in `https://www.w3.org/TR/owl2-primer/`, translate the sentences you received from another group into OWL axioms.
- Write the sentences and corresponding OWL axioms here: `http://piratepad.net/YszSBfmTNY`

Example Sentences I

- Topic: Cricket
  - Cricket is a team sport
  - Cricket can be played between two teams
  - Sachin Tendulkar is a cricketer with the most number of runs in a world cup
  - There are three formats in playing cricket game
  - Cricket is the worlds second most popular sport
  - Cricket has 11 members in each team.
  - International Cricket Council (ICC) is the international governing body of cricket.
  - Cricket was first played in 18th century.

Example Sentences II

- Topic: Baseball
  - Each game has at least 9 innings
  - Any baseball player must belong to exactly one baseball team
  - Any regular season game has both a home team and away team
  - A player's batting average must be between 0 and 1

Example Sentences III

- Topic: Presidential Campaign Financing
  - Candidates can collect donation from donors.
  - Every donor has occupation, address, amount of donation, date and time for donation.
  - Every contributor has at least one candidate to donate
  - One donator can donate to more than one candidate
  - Every contribution has a transaction id
  - There is a limit on the total amount of donation collected from each state.

Examples (and non-examples) from Cricket domain I

Cricket is a team sport

```
ClassAssertion( :TeamSport :Cricket )
```

Cricket can be played between two teams (Cricket is played by exactly two teams)

```
SubClassOf( :CricketGame
            ObjectExactCardinality( 2 :playedBy :CricketTeam ))
```

Sachin Tendulkar is a cricketer with the most number of runs in a world cup.
$\rightsquigarrow$ *Not truly expressible in OWL*

There are three formats in playing cricket game. (Every cricket game has three possible formats.)

```
SubClassOf(
    :CricketGame
    ObjectSomeValuesFrom(
        :hasFormat
        ObjectOneOf( :format1 :format2 :format3 ) ) )

DifferentIndividuals( :format1 :format2 :format3 )
```

Examples (and non-examples) from Cricket domain II

Cricket is the world's second most popular sport.
$\rightsquigarrow$ *Not truly expressible in OWL*

Cricket has 11 members in each team. (Every cricket team has 11 players).

```
SubClassOf( :CricketTeam
      ObjectExactCardinality( 11 :hasPlayer :CricketPlayer ) )
```

International Cricket Council (ICC) is the international governing body of cricket.

```
ObjectPropertyAssertion( :governingBodyOf :ICC :Cricket )
```

Cricket was first played in 18th century.

```
DataPropertyAssertion(
      :wasFirstPlayed
      :Cricket
      "1701-01-01T00:00:00"^^xsd:dateTime )
```

Examples from Baseball domain I

Each game has at least 9 innings.

```
SubClassOf( :BaseballGame
            ObjectMinCardinality( 9 :hasInning :Inning ) )
```

Any baseball player must belong to exactly one baseball team.

```
SubClassOf( :BaseballPlayer
            ObjectExactCardinality( 1 :belongsTo :BaseballTeam ) )
```

Any regular season game has both a home team and away team.

```
SubClassOf(
   :RegularSeasonBaseballGame
   ObjectIntersectionOf(
      ObjectSomeValuesFrom( :hasHomeTeam :BaseballTeam )
      ObjectSomeValuesFrom( :hasAwayTeam :BaseballTeam ) ) )
```

Examples from Baseball domain II

A player's batting average must be between 0 and 1.

```
SubClassOf(
   :BaseballPlayer
   ObjectIntersectionOf(
      DataSomeValuesFrom(
         :hasBattingAverage
         DatatypeRestriction( xsd:decimal
                              xsd:minInclusive "0"^^xsd:decimal
                              xsd:maxInclusive "1"^^xsd:decimal )
```

Every donor has occupation, address, amount of donation, date and time for donation.

```
SubClassOf(
  :Donor
  ObjectIntersectionOf(
    ObjectSomeValuesFrom( :hasOccupation :Occupation )
    DataSomeValuesFrom( :hasAddress xsd:string )
    DataSomeValuesFrom( :hasDonationDateTime xsd:dateTime ) ) )
```

Candidates can collect donation from donors.

$\rightsquigarrow$ *Candidates collects donation from zero or more donors.*

$\rightsquigarrow$ *equivalent to universal truth/tautology.*

$\rightsquigarrow$ *No axiom is needed.*

Every contributor (donor) has at least one candidate to donate.

```
SubClassOf(
  :Donor
  ObjectSomeValuesFrom( :donatesTo :PresidentialCandidate ) )
```

One donor can donate to more than one candidate.
$\rightsquigarrow$ *is it intended to be the same as the previous statement?*

Every contribution (donation) has a transaction id.

```
SubClassOf( :Donation
            DataSomeValuesFrom( :hasTransactionID xsd:string ) )
```

There is a limit on the total amount of donation collected from each state.

```
SubClassOf( :State
  DataSomeValuesFrom ( :hasTotalCollectedDonation xsd:nonNegativeInteg
```

Remarks/Caveat

- Some of the sentences cannot be fully captured in OWL because OWL is not expressive enough.
- Some sentences may need domain and range restrictions of properties to completely describe the intent.
- The axioms given earlier may not reflect a good modeling practice; they are just examples of the use of OWL constructs.
  - If we wish to obtain a good ontology, we need to do it more carefully; the axioms may look completely different than the ones described earlier.

OWL Tools

- Protégé – ontology editor (free)
  - Let's try to see different serializations of some of the earlier examples in Protégé.
- TopBraid Composer – ontology editor (commercial)
- OWL API (free Java library)
- OWL Syntax converter (running on top of, rather outdated, OWL API) – `http://mowl-power.cs.man.ac.uk:8080/converter/`